Creating Secure iPhone and Mac OS X Applications

Session 503

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

Security Hardening

Geoff Keating
Product Security Engineer

Security Hardening in Snow Leopard

In this part of the presentation:

- Hardening features enabled by default
  - What do they mean for you?
- Hardening features you should turn on

Later:

- Determining exploitability of crashes
  - Automatically or
  - By hand

Hardening Features on by Default

Where?

- Target 10.5 or higher
- Build 64-bit

- Available for C, C++, Objective-C, Objective-C++

Hardening Features on by Default

Effects

- Minimal performance impact (too small to measure)
- All together: Very small code size impact
- Just hardening
  - Won’t save you if it all goes really wrong

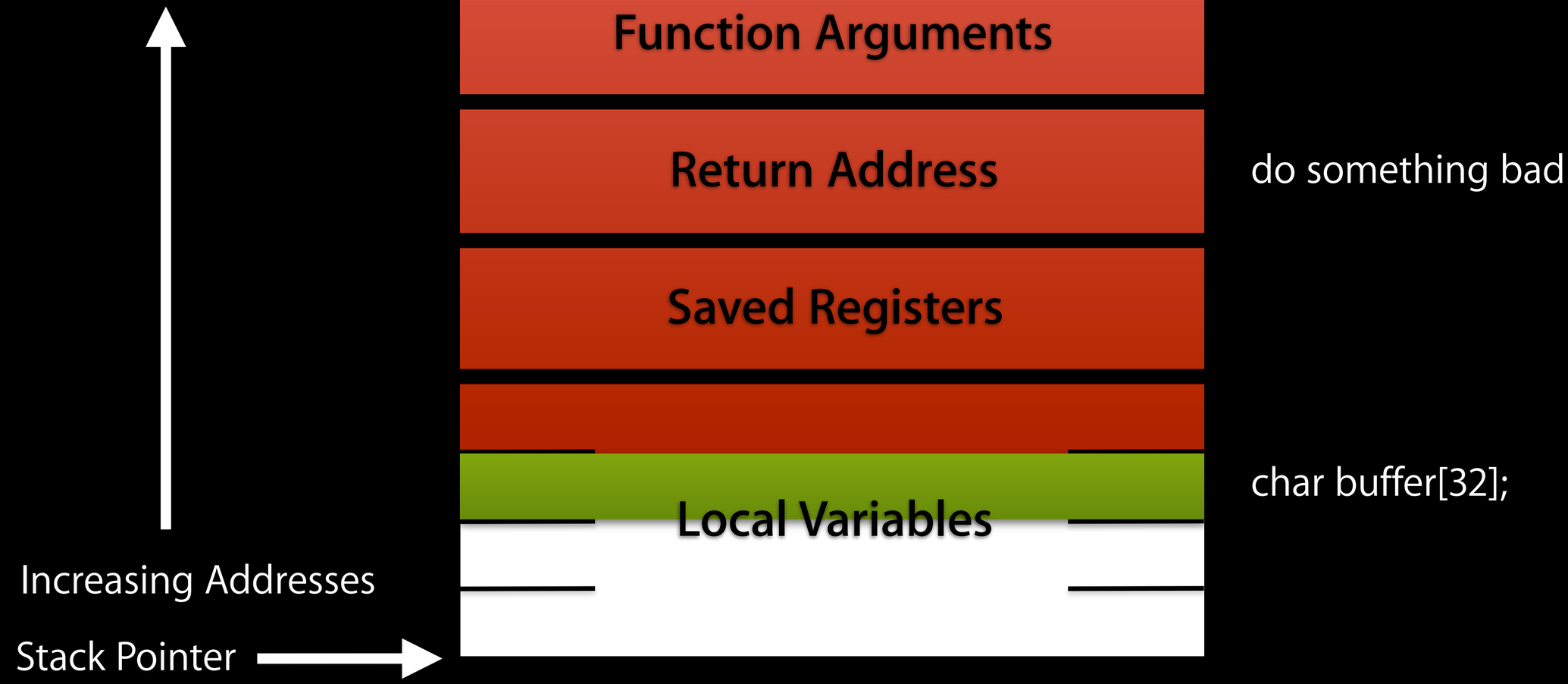
Stack Overflow

Simple example

```
void print_upper(char * from_network, size_t length)
{
  char buffer[32];
  for (int i = 0; i < length; i++)
    buffer[i] = toupper(from_network[i]);
  buffer[length] = 0;
  puts(buffer);
}
```

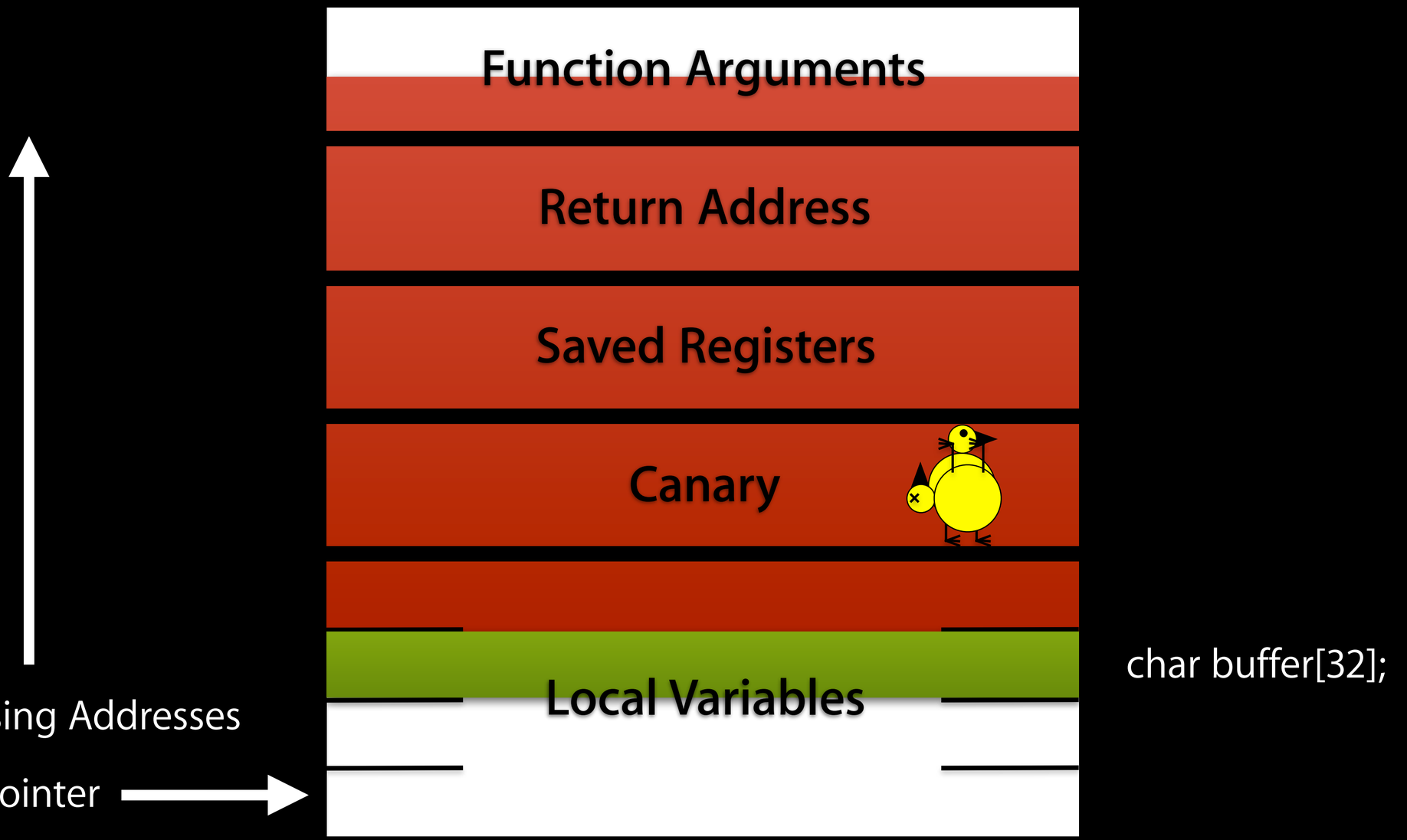
Stack Overflow

Stack frame



Stack Overflow

Stack frame protection with a ‘canary’



Stack Overflow

Crash report

```
0   libSystem.B.dylib               0x00007fff86be7882 __kill + 10
1   libSystem.B.dylib               0x00007fff86c6ad17 __abort + 103
2   libSystem.B.dylib               0x00007fff86c4f83c release_file_streams_for_task + 0
3   brokendaemon                    0x0000000100000e3b print_upper + 127
4   brokendaemon                    0x0000000100000e89 main + 50
5   brokendaemon                    0x0000000100000db4 start + 52
```

```
$ nm -arch x86_64 -n /usr/lib/libSystem.B.dylib | grep -B1 release_file_streams_for_task
00000000000d07e9 T ___stack_chk_fail
00000000000d083c t _release_file_streams_for_task
```

Stack Overflow

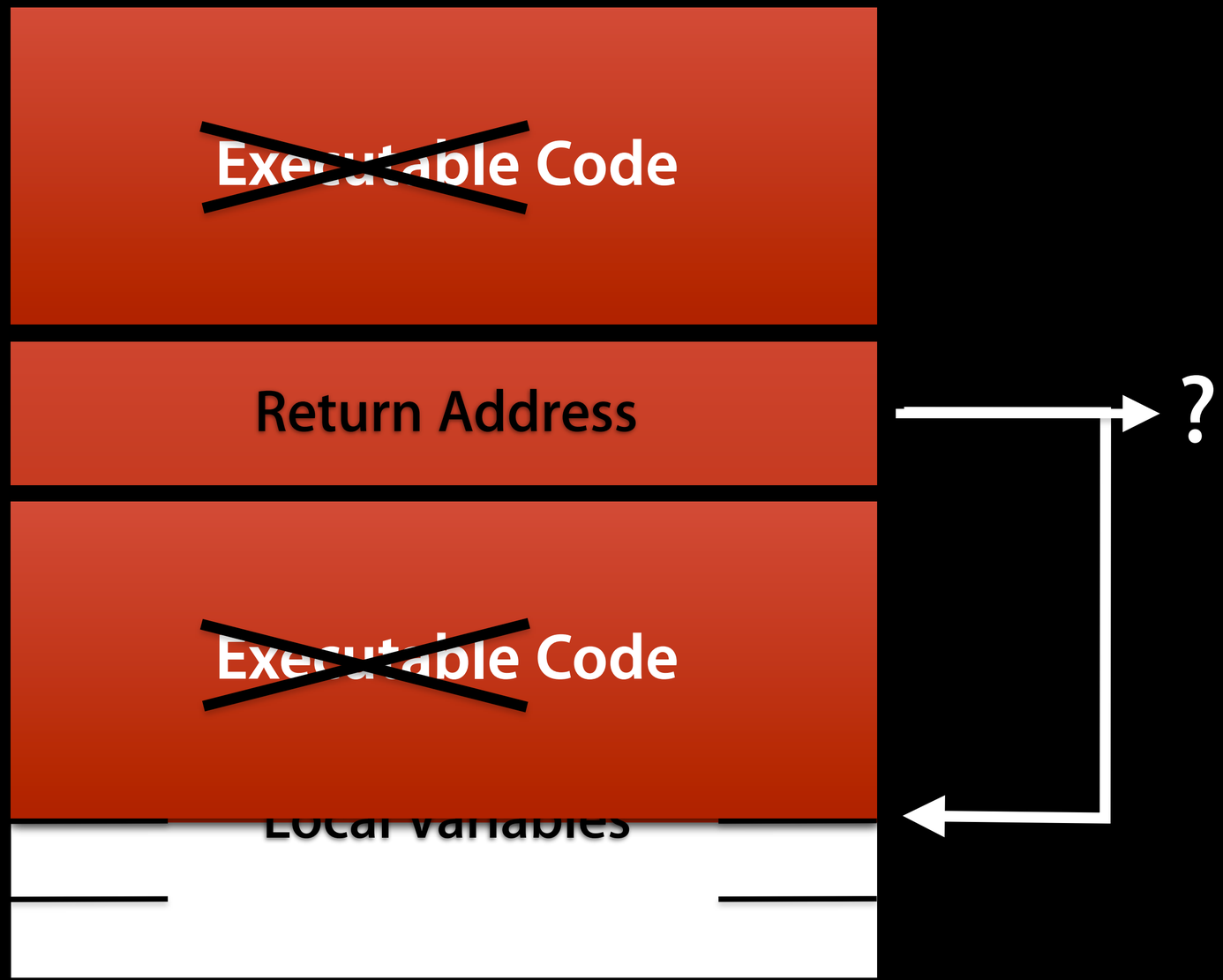
More!

Setting	Value
GCC 4.2 – Language	
Other C Flags	-fstack-protector-all

- Increased performance impact
- Increased security

Code Injection

Exploiting a vulnerability



Code Injection

"No Execute" for data

- On Mac OS X
  - Applies to stack, static data, and heap
    - Stack: 64-bit PPC, 32-bit Intel, and 64-bit Intel
    - Static data and heap: 64-bit PPC and 64-bit Intel
    - `mprotect()` may be used to make data executable when necessary
- On iPhone
  - `mprotect()` may not be used
  - All code must also be signed
  - No interpreters allowed (except system JavaScript)

Misuse of Library Functions

```
char buffer[1024];
strcpy (buffer, untrusted_data);
```

```
char buffer[1024];
strlcpy (buffer, untrusted_data, sizeof (buffer));
```

```
char buffer[1024];
if (strlcpy (buffer, untrusted_data, sizeof (buffer))
    >= sizeof (buffer))
  /* handle error */
```

“_FORTIFY_SOURCE”

- Compiler tries to determine array sizes
- Compiler replaces standard library functions
  - `strcpy` → `__strcpy_chk`
- Typical crash report:

```
0   libSystem.B.dylib           0x00007fff86be7882 __kill + 10
1   libSystem.B.dylib           0x00007fff86c6ad17 __abort + 103
2   libSystem.B.dylib           0x00007fff86c60235 __chk_fail + 105
3   libSystem.B.dylib           0x00007fff86bcd44a __strcpy_chk + 47
4   hellobuf                    0x0000000100000ecc main + 68
```

Extra Heap Consistency Checks

Administrator vs. Root

- First account created is an administrator
- Many systems have just one account

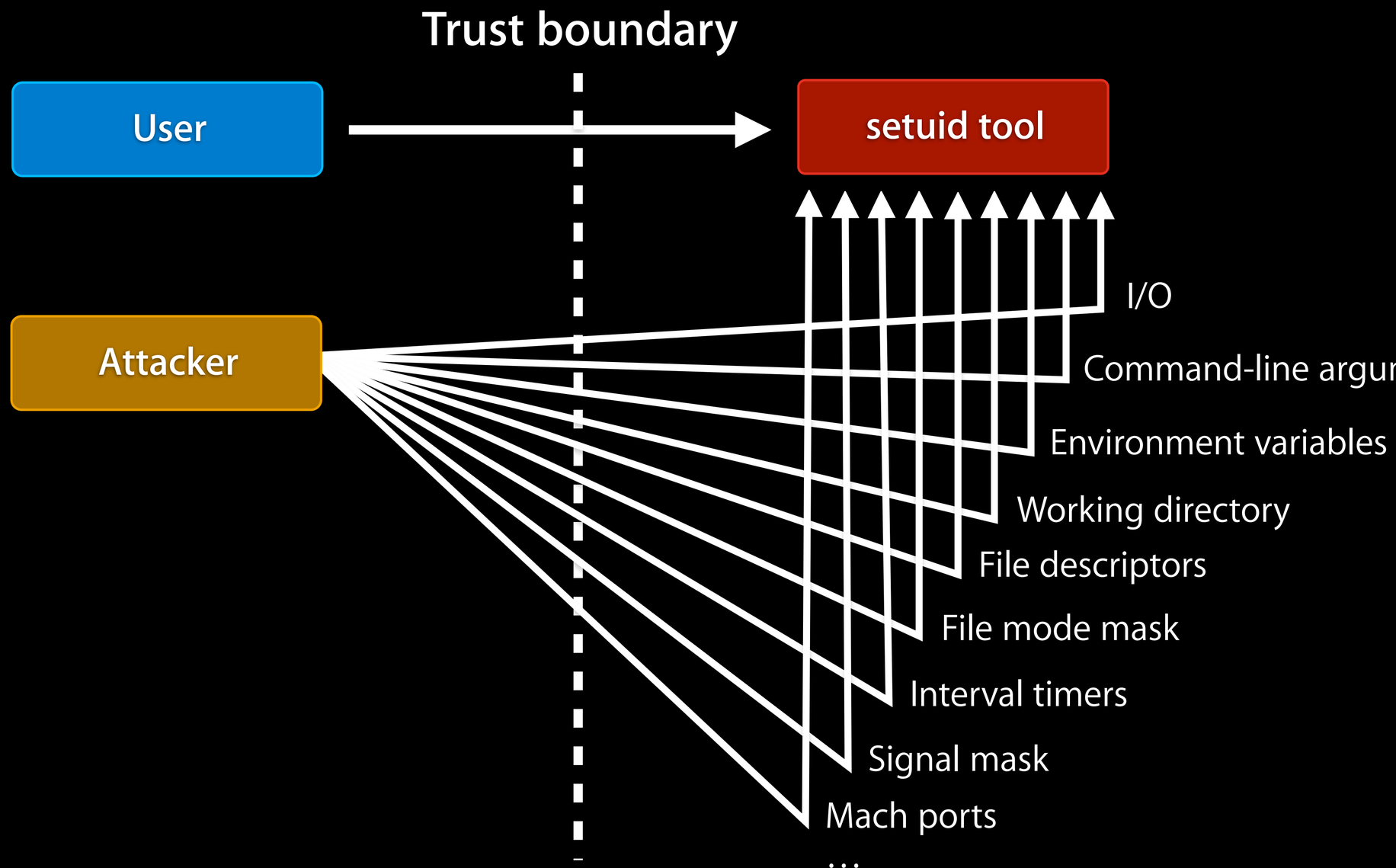
- Goal: Administrators have no extra privileges
  - Except that they can use their password to become root

- Actual: Many system directories are writable by administrators

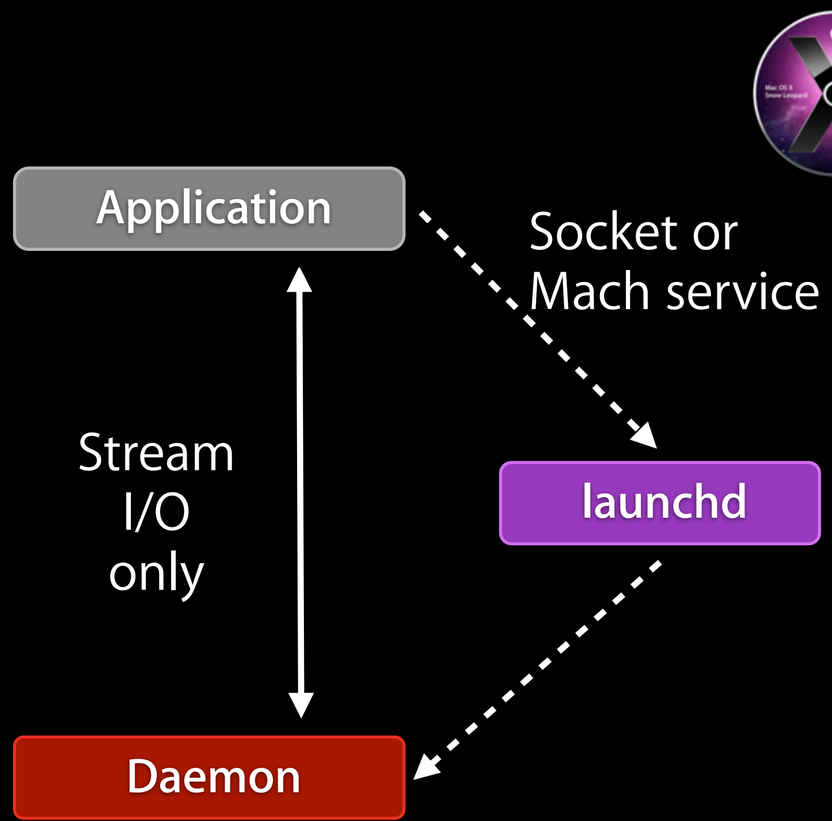
Permissions Changes

- Some directories are no longer writeable by administrators
  - Most system applications
  - /Library/ScriptingAdditions
- More similar changes coming

`setuid` Programs



Much Less `setuid`

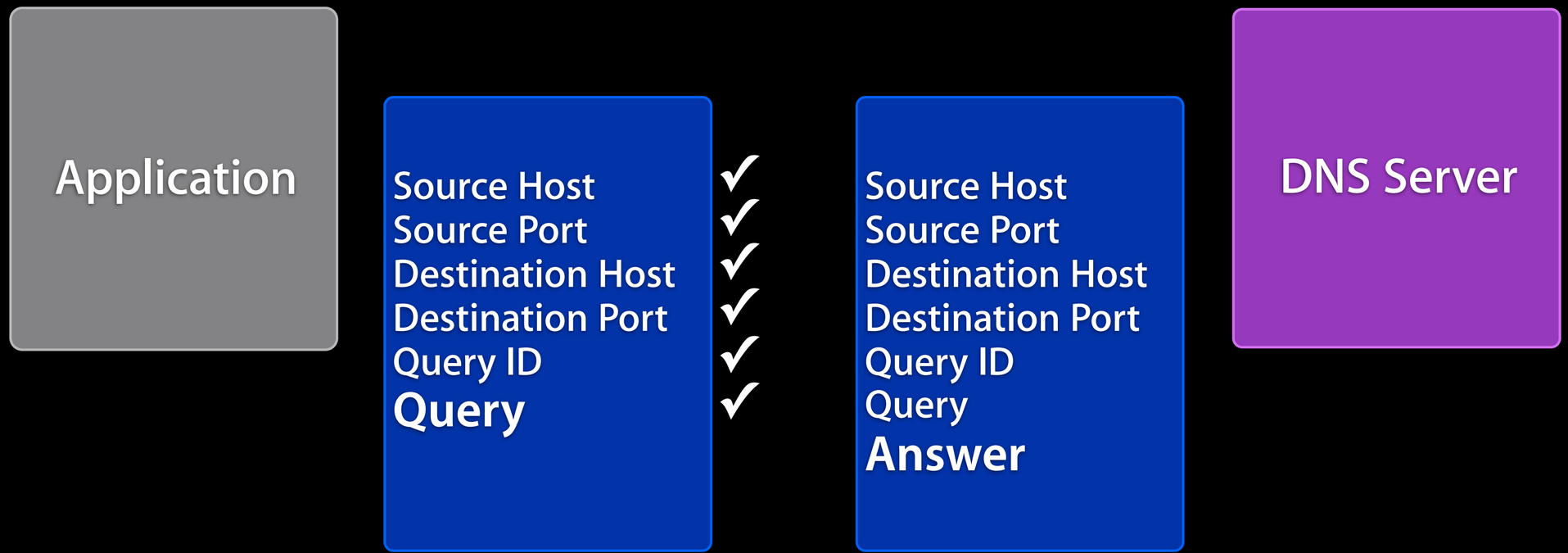


- Many system applications are no longer using set-user-ID
- AppKit will refuse to run `setuid` and ask for an admin password
- Your executables should not be `setuid`

- It can be done: iPhone has no `setuid` programs!

Domain Name System Attack

Normal operation



Domain Name System Attack

DNS spoofing attack

Application

✓
?
✓
✓
2^{16}
✓

Source Host
Source Port
Destination Host
Destination Port
Query ID
Query
Answer

DNS Server

Attacker

Source Host
Source Port
Destination Host
Destination Port
Query ID
Query
Bad Answer

UDP Port Randomization

- This attack:
  - DNS ports randomized in Mac OS X 10.5.x and Security Update 2008–09
  - Now the attacker must send one billion packets on average
  - Eventually DNSSEC will be deployed and completely prevent attack
- What about other protocols?
  - UDP ports randomized on Snow Leopard for all protocols
- Potential gotcha:
  - Ports will be reused faster than previously

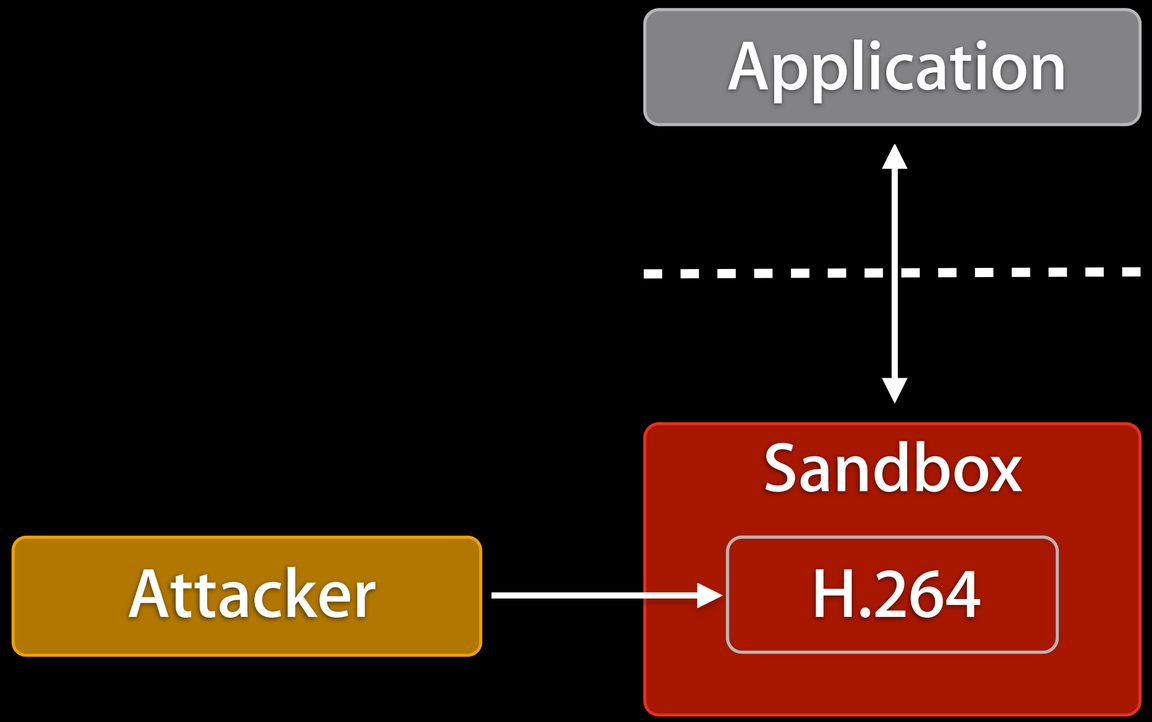
Code Layout Randomization

- System libraries: Randomized since Leopard
- Executables should be built as PIE to be more randomized
- Some system executables adopting PIE in Snow Leopard

Setting	Value
▼Linking	
Other Linker Flags	-fpie
▼GCC 4.2 - Code Generation	
Generate Position-Dependent Code	☐
▼GCC 4.2 - Language	
Other C Flags	-fpie

Sandboxing

Sandbox



- Use `sandbox_init` API
- Five sandboxes available:
  - No internet protocols
  - No sockets at all
  - No filesystem writes
  - No filesystem writes except temporary files
  - Pure computation only
- Custom sandboxes not yet available

Crash Exploitability Triage

Drew Yao
Product Security Engineer

Topics

- Determining exploitability
- Automated
  - CrashWrangler
- Manual
  - Crash logs
  - Debugger

Motivation: Fuzzing

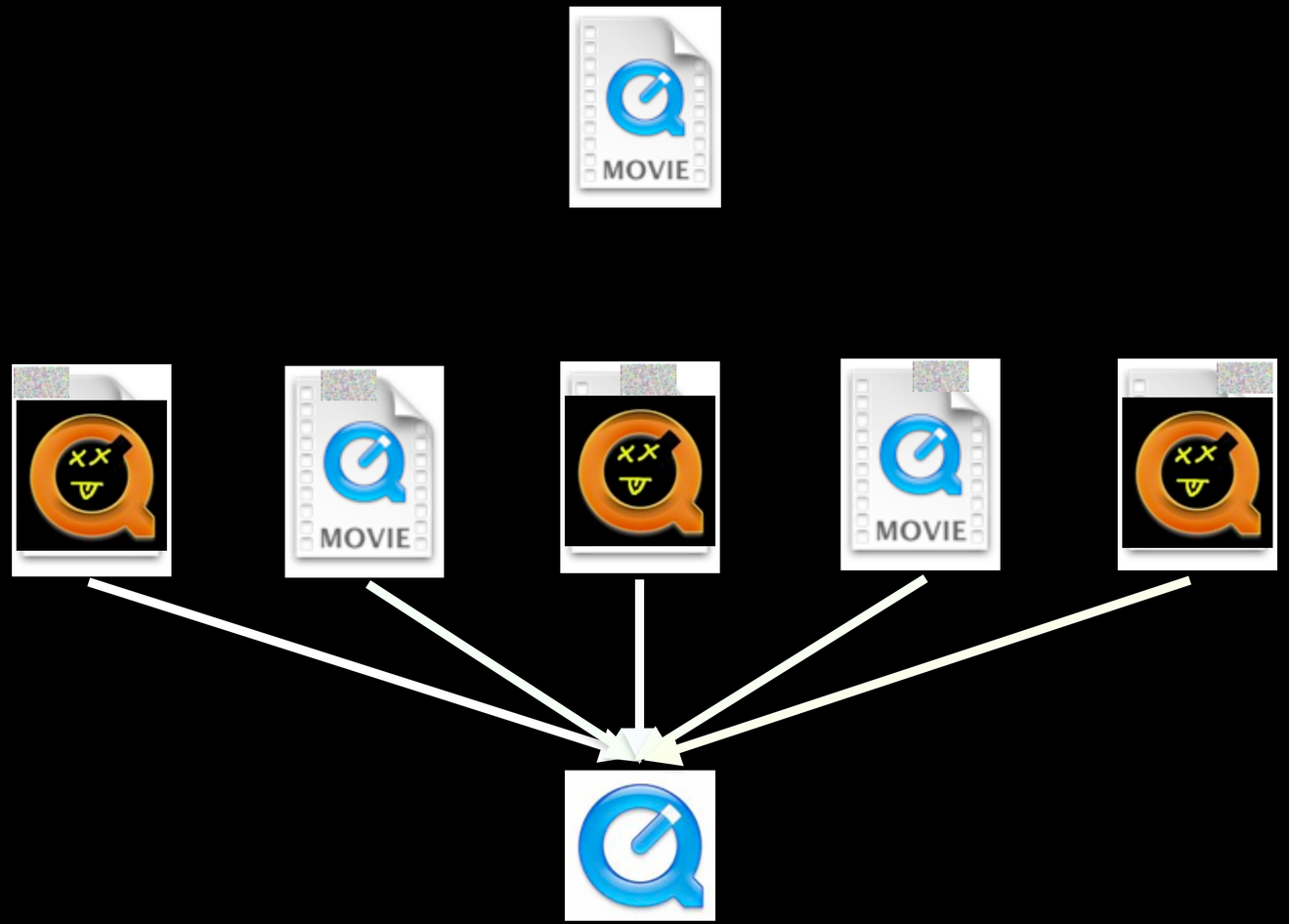
Fuzzing

- Use automation to
  - Generate malformed input
  - Feed it to the app
- Trigger crash

Why Fuzzing

- A primary tool for hackers
  - No access to source code required
  - Tends to find exploitable flaws
  - Relatively easy to create and run
- Attackers are actively fuzzing commercial products

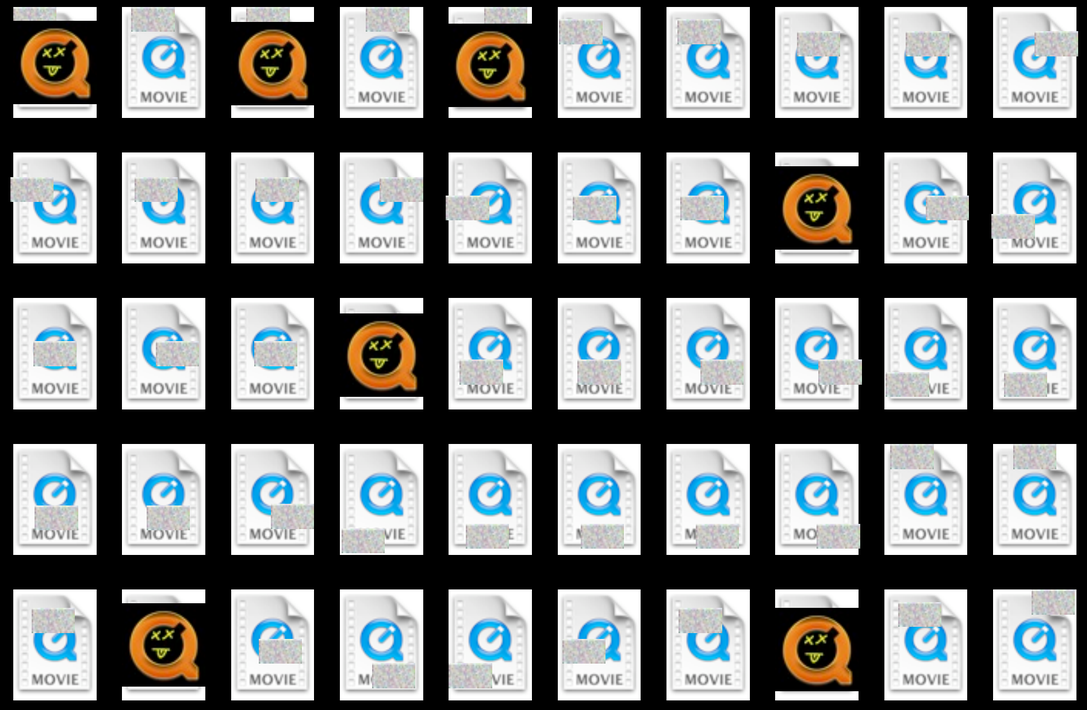
Fuzzing



Fuzzing Challenges

- Which are duplicates?
- Which are exploitable?

Scale



Introducing CrashWrangler

CrashWrangler

- Runs on Mac OS X 10.5 and 10.6
- Automatically determines:
  - Exploitability of crash
  - Whether or not crash is duplicate
- Heuristics
- Will be made generally available

CrashWrangler

- Mach exception handler
  - Fork and exec process
  - If child crashes, examine it and output a log file
- Script to analyze logs to bucket duplicates

CrashWrangler Used For:

- Fuzzing
- Screening crash reports
  - Need to prioritize quickly
  - No time to debug
  - Division of labor

Demo

Manual Crash Triage

Exploitability

- Focusing on arbitrary code execution
  - Not considering denial of service, information disclosure, etc.

- Trusted input -> not exploitable

Exploitable Crashes: Memory Corruption

- Writing where you aren't supposed to write
  - Past end of buffer
  - To arbitrary or wild pointer
  - To freed memory
- Often allows the attacker to alter control flow

Memory Corruption == Exploitable

- Not just controlled buffer overflows
  - Very large buffer overflows
  - Byte swapping after buffer
  - Bzero after buffer
  - Use after free

Memory Corruption
Probable or Certain
Very Improbable
Improbable

Stack Corruption

```
Exception Type:  EXC_BAD_ACCESS (SIGSEGV)
Exception Codes: KERN_INVALID_ADDRESS at 0x0000000041414141
Crashed Thread:  0  Dispatch queue: com.apple.main-thread

Thread 0 Crashed:  Dispatch queue: com.apple.main-thread
0   ???                                 0x41414141 0 + 1094795585
```

Illegal Instruction

```
Exception Type:  EXC_BAD_INSTRUCTION (SIGILL)
Exception Codes: 0x0000000000000001,
0x0000000000000000
```

```
(gdb)
Program received signal
EXC_BAD_INSTRUCTION, Illegal
instruction/operand.
```

Abort: -fstack-protector

```
Exception Type:  EXC_BAD_ACCESS (SIGSEGV)
Exception Codes: KERN_INVALID_ADDRESS at 0x0000000041414140
Crashed Thread:  0  Dispatch queue: com.apple.main-thread

Thread 0 Crashed:  Dispatch queue: com.apple.main-thread
0   libSystem.B.dylib               0x90549a70 strlen + 16
1   libSystem.B.dylib               0x9055baf5 strdup + 26
2   libSystem.B.dylib               0x90599966 asl_set_query + 324
3   libSystem.B.dylib               0x9059981f asl_set + 39
4   libSystem.B.dylib               0x90598b27 vsyslog + 495
5   libSystem.B.dylib               0x90598936 syslog + 34
6   libSystem.B.dylib               0x90627396 __stack_chk_fail + 93
7   a.out                           0x00001f6d main + 77
8   ???                             0x41414141 0 + 1094795585
```

Abort: -fstack-protector

```
Application Specific Information:
__abort() called

Thread 0 Crashed:  Dispatch queue: com.apple.main-thread
0   libSystem.B.dylib               0x905b23ca __kill + 10
1   libSystem.B.dylib               0x905b23be kill$UNIX2003 + 32
2   libSystem.B.dylib               0x9062e7cf raise + 26
3   libSystem.B.dylib               0x90644b61 __abort + 124
4   libSystem.B.dylib               0x9062739b release_file_streams_for_task + 0
5   stack_buffer_overflow           0x00001f6b main + 81
6   ???                             0x41414141 0 + 1094795585
```

```
Snow Leopard:
<__stack_chk_fail+93>:              call   0x945774a6 <__abort>
<release_file_streams_for_task>:  push   %ebp

Leopard:
<__stack_chk_fail+86>:    call   0x924a55c1 <__abort>
<append_int>:           push   %ebp
```

Abort - _FORTIFY_SOURCE

```
Exception Type:  EXC_CRASH (SIGABRT)
Exception Codes: 0x0000000000000000, 0x0000000000000000
Crashed Thread:  0  Dispatch queue: com.apple.main-thread

Application Specific Information:
__abort() called

Thread 0 Crashed:  Dispatch queue: com.apple.main-thread
0   libSystem.B.dylib               0x905b23ca __kill + 10
1   libSystem.B.dylib               0x905b23be kill$UNIX2003 + 32
2   libSystem.B.dylib               0x9062e7cf raise + 26
3   libSystem.B.dylib               0x90644b61 __abort + 124
4   libSystem.B.dylib               0x90639af1 __chk_fail + 118
5   libSystem.B.dylib               0x905eacfb __memset_chk + 25
6   fortify_source_overflow         0x00001f7f main + 41
7   fortify_source_overflow         0x00001f4d start + 53
```

Abort - MallocCorruptionAbort

```
Application Specific Information:
*** error for object 0x1027f4: incorrect checksum for freed object - object was
probably modified after being freed.

Thread 0 Crashed:  Dispatch queue: com.apple.main-thread
0   libSystem.B.dylib               0x905b23ca __kill + 10
1   libSystem.B.dylib               0x905b23be kill$UNIX2003 + 32
2   libSystem.B.dylib               0x9062e7cf raise + 26
3   libSystem.B.dylib               0x90644bcc abort + 93
4   libSystem.B.dylib               0x906333d0 szone_error + 343
5   libSystem.B.dylib               0x9063350b free_list_checksum_botch + 51
6   libSystem.B.dylib               0x9054b8b8 tiny_malloc_from_free_list + 108
7   libSystem.B.dylib               0x9054aae3 szone_malloc_should_clear + 297
8   libSystem.B.dylib               0x9054a968 malloc_zone_malloc + 81
9   libSystem.B.dylib               0x90548a38 malloc + 50
10  a.out                           0x00001f47 main + 149
11  a.out                           0x00001ea9 start + 53
```

Crash in malloc/free/objc_MsgSend

```
Exception Type:  EXC_BAD_ACCESS (SIGSEGV)
Exception Codes: KERN_INVALID_ADDRESS at 0x0000000fffffff0
Crashed Thread:  0

Thread 0 Crashed:
0   libSystem.B.dylib               0x923c1338 tiny_malloc_from_free_list + 62
1   libSystem.B.dylib               0x923ba1cd szone_malloc + 180
2   libSystem.B.dylib               0x923ba0d8 malloc_zone_malloc + 81
3   libSystem.B.dylib               0x923ba06c malloc + 55
4   a.out                           0x00001fdf main + 153
5   a.out                           0x00001f1a start + 54
```

Crash in malloc/free/objc_MsgSend

```
Exception Type:  EXC_BAD_ACCESS (SIGBUS)
Exception Codes: KERN_PROTECTION_FAILURE at 0x00000000ff505326
Crashed Thread:  0

Application Specific Information:
objc_msgSend() selector name: release


Thread 0 Crashed:
0   libobjc.A.dylib                 0x90a46f67 objc_msgSend + 23
1   com.apple.someframework         0x922eb788 somefunction + 248
```

Write to Invalid Address

```
Program received signal EXC_BAD_ACCESS, Could not access
memory.
Reason: KERN_INVALID_ADDRESS at address: 0x61616161
0x00001ffa in main ()
(gdb) x/i $pc
0x1ffa <main+16>:    movb   $0x0,(%eax)
```

Executing Invalid Address

```
Exception Type:  EXC_BAD_ACCESS (SIGSEGV)
Exception Codes: KERN_INVALID_ADDRESS at 0x0000000051515151
Crashed Thread:  0  Dispatch queue: com.apple.main-thread

Thread 0 Crashed:  Dispatch queue: com.apple.main-thread
0   ???                                 0x51515151 0 + 1364283729
1   crashexec                           0x00001f75 start + 53
```

Calling Invalid Address

```
Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x7777777f
0x00001ff9 in main ()
(gdb) x/i $pc
0x1ff9 <main+11>:    call   *0x8(%eax)
```

Different Crash Each Run

- Memory layout slightly different each time
- Memory corruption leads to different crashes
- Try running with libgmalloc(3)

Memory Corruption

Probable or Certain

Very Improbable

Improbable

Divide by Zero

```
Exception Type:  EXC_ARITHMETIC (SIGFPE)
Exception Codes: EXC_I386_DIV (divide by zero)
```

```
(gdb)
Program received signal EXC_ARITHMETIC, Arithmetic exception.
```

Recursion Stack Overflow

```
Exception Type:  EXC_BAD_ACCESS (SIGSEGV)
Exception Codes: KERN_PROTECTION_FAILURE at 0x00000000bf7ffffc
Crashed Thread:  0  Dispatch queue: com.apple.main-thread

Thread 0 Crashed:  Dispatch queue: com.apple.main-thread
0   recursion                          0x00001f80 foo + 6
1   recursion                          0x00001f85 foo + 11
2   recursion                          0x00001f85 foo + 11
3   recursion                          0x00001f85 foo + 11
4   recursion                          0x00001f85 foo + 11
5   recursion                          0x00001f85 foo + 11
6   recursion                          0x00001f85 foo + 11
7   recursion                          0x00001f85 foo + 11
8   recursion                          0x00001f85 foo + 11
9   recursion                          0x00001f85 foo + 11
10  recursion                          0x00001f85 foo + 11
11  recursion                          0x00001f85 foo + 11
12  recursion                          0x00001f85 foo + 11
13  recursion                          0x00001f85 foo + 11
14  recursion                          0x00001f85 foo + 11
15  recursion                          0x00001f85 foo + 11
...
```

Recursion Stack Overflow

```
Program received signal EXC_BAD_ACCESS, Could
not access memory.
Reason: KERN_PROTECTION_FAILURE at address:
0xbf7ffffc
0x00001fec in foo ()
(gdb) x/i $pc
0x1fec <foo+6>:      call   0x1fe6 <foo>
```

Recursion Stack Overflow

Not Exploitable

```
Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_PROTECTION_FAILURE at address: 0xbf7ffffc
0x00001fec in foo ()
(gdb) x/i $pc
0x1fec <foo+6>:      call   0x1fe6 <foo>
```

Exploitable

```
Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x7777777f
0x00001ff9 in main ()
(gdb) x/i $pc
0x1ff9 <main+11>:    call   *0x8(%eax)
```

Abort/Assertion

```
Exception Type:  EXC_CRASH (SIGABRT)
Exception Codes: 0x0000000000000000, 0x0000000000000000
Crashed Thread:  0  Dispatch queue: com.apple.main-thread

Application Specific Information:
Assertion failed: (n < 10), function main, file test.c, line 5.


Thread 0 Crashed:  Dispatch queue: com.apple.main-thread
0   libSystem.B.dylib               0x905b23ca __kill + 10
1   libSystem.B.dylib               0x905b23be kill$UNIX2003 + 32
2   libSystem.B.dylib               0x9062e7cf raise + 26
3   libSystem.B.dylib               0x90644bcc abort + 93
4   libSystem.B.dylib               0x9063180e __assert_rtn + 252
5   a.out                           0x00001f6a main + 76
6   a.out                           0x00001f15 start + 53
```

Memory Corruption

Probable or Certain

Very Improbable

Improbable

Null Dereference

```
Exception Type:  EXC_BAD_ACCESS (SIGBUS)
Exception Codes: KERN_PROTECTION_FAILURE at 0x0000000000000000
```

Null Dereference

- Depends if offset could potentially reach outside of null page

```
char *buf = NULL;

//potentially exploitable
buf[evil_num2] = evil_data;

//not exploitable
buf[0] = evil_data;

//probably not exploitable
((struct foo*)buf->member = evil_data;
```

Read Out of Bounds

```
Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x41414141
0x00001ff6 in main ()
(gdb) x/i $pc
0x1ff6 <main+16>:    movzbl (%eax),%eax
```

Read Out of Bounds

- Potentially exploitable
  - Further investigation
  - Run again with libgmalloc
- Potential information disclosure
- Overread followed by overwrite is exploitable

```
int *buf = malloc(size);

for (i=0;i<evil_num;i++) {
    buf[i] = byteswap(buf[i]);
}
```

CrashWrangler vs. Manual

	CrashWrangler	Manual assessment
Detect stack corruption	✓	✓
Detect recursion	✓	✓
Analyze exception type	✓	✓
Analyze stack trace	✓	✓
Determine access type (RWX)	✓	✓
Examine access address	✓	✓
Determine trust level of input	✗	✓
Diagnose security issues other than memory corruption	✗	✓

Memory Tools

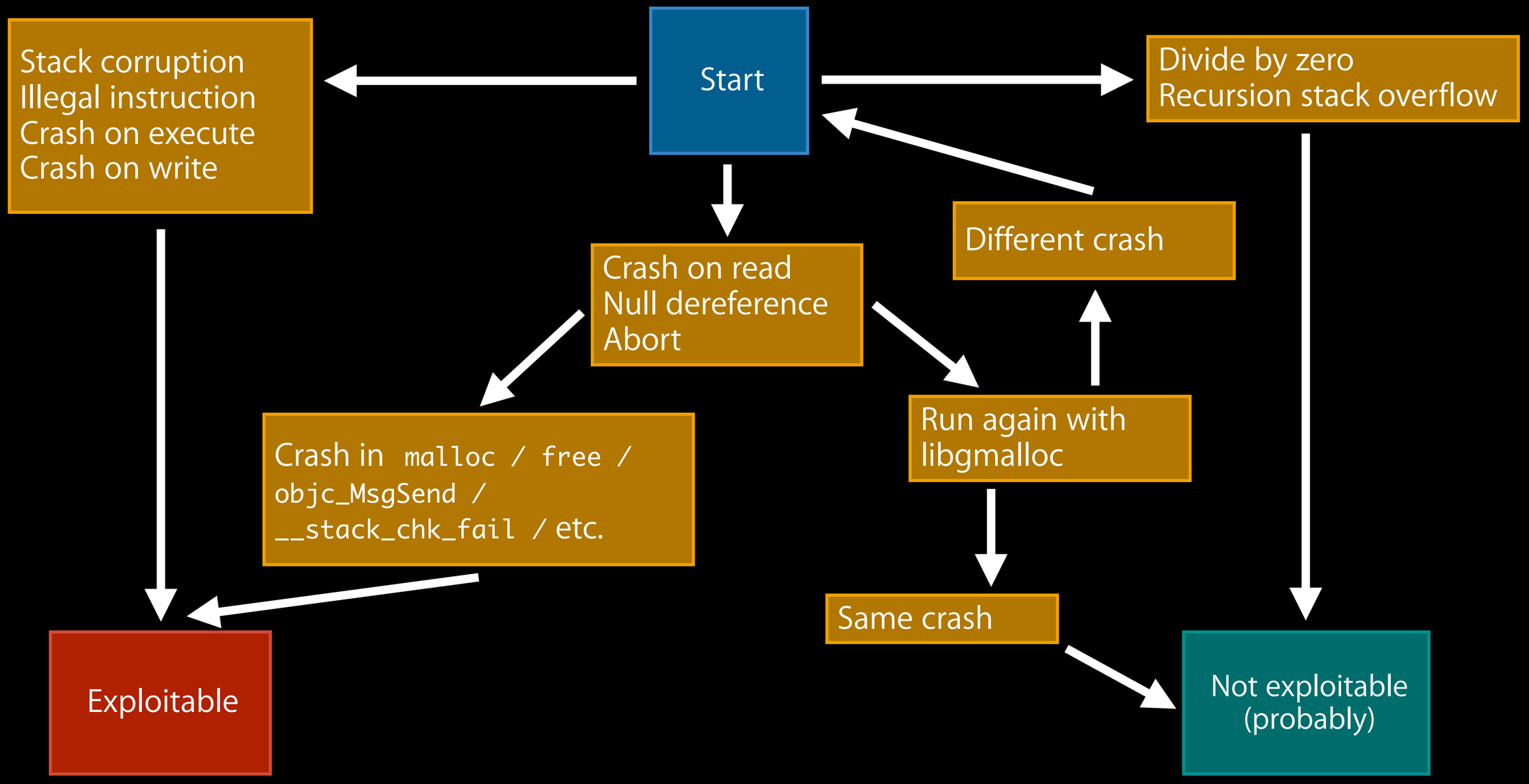
malloc(3) Environment Variables

- MallocScribble
  - Detect use of uninitialized heap memory or freed memory
- MallocGuardEdges
  - Detect buffer overflows of large buffers
- MallocCorruptionAbort
  - Detect heap corruption (SnowLeopard only, on by default for 64-bit)

libgmalloc(3)

- Guard page after each allocation
- Unmap on free
- Crash immediately when accessing freed memory or memory after heap buffer
- Extremely slow
- MALLOC_ALLOW_READS

Exploitability Flow Chart



Crash Triage Summary

- Memory corruption bugs considered exploitable
  - Hard to prove not exploitable
- Detect memory corruption via crash log, debugger, CrashWrangler
- CrashWrangler used for fuzzing or triaging crashes
- Use `malloc` tools and hardening build flags

More Information

Craig Keithley
Technology Evangelist
keithley@apple.com

CrashWrangler
http://developer.apple.com/wwdc/attendee/sessions/session.php?id=503

Apple Product Security
product-security@apple.com
http://bugreport.apple.com

Secure Coding Guide
http://developer.apple.com/documentation/Security/Conceptual/SecureCodingGuide/index.html

Labs

Mac OS X Security Lab	OS X Lab A Wednesday 12:00–3:15 PM
iPhone OS Security Lab	iPhone Lab D Thursday 5:00–6:15 PM

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09